Preprint
UCRL-JC-139907

Experiences Applying Meta-Data to Bioinformatics

T. Critchlow, R. Musick, T. Slezak

This article was published in
Information Sciences, Vol 139 (1-2) 2001
Elsevier Science Inc.

November, 2001

Experiences Applying Meta-Data to Bioinformatics

Terence Critchlow[1], Ron Musick[2], Tom Slezak[3]
critchlow1@llnl.gov
7000 East Ave. MS L-560 Livermore CA 94550[1]
Center For Applied Scientific Computing[1]
Biology and Biotechnology Research Program[3]
Lawrence Livermore National Laboratory
iKuni Inc[2]

Abstract

Bioinformatics is facing the daunting challenge of providing geneticists and biologists effective, efficient access to data currently distributed among dynamic, heterogeneous data sources. Complicating the problem is the speed at which the underlying science and technology evolve, leaving the terminology, databases and interfaces to catch up. As the genomics community moves from sequences to functional genomics, the pressure to find a solution is increasing. Realistically addressing this problem, whether through a data warehouse, multi-database, federated database, or other approach, requires development of an scalable, flexible infrastructure that can quickly adapt to meet user needs in this extremely dynamic environment. This is best accomplished by extensively using meta-data to reduce the application's maintenance costs. Using the DataFoundry project as an example, this paper discusses the first steps and practical problems of developing a meta-data-based infrastructure capable of meeting the demands of an active scientific community. It also demonstrates how much bioinformatics must still progress before it can truly satisfy its users.

1. Introduction

In biology, as in most sciences, progress is made through a hypothesize-and-test cycle — a scientist examines the current set of knowledge, identifies something that cannot be adequately explained, forms an educated hypothesis about why it occurs, designs and performs an experiment to test the hypothesis, and interprets the experiment's results. Because each experiment may consume considerable resources, it is important that the hypothesis being tested is based on a thorough understanding of the work it is building on. This helps maximize the overall scientific value of the work by ensuring that the hypothesis being tested has not been previously evaluated (at least in the same way), does not contradict the results of previous experiments, and can explain a poorly understood behavior.

Success in formulating hypotheses that meet the above criteria is based, in large part, on the scientists' ability to efficiently obtain and understand all of the relevant information in a timely fashion. With data spread throughout multiple locations — each using a different data format, user interface, and terminology — significant time may be wasted trying to identify and obtain enough information to form a reasonable hypothesis. Worse, the scientist may have overlooked relevant information contained in unfamiliar data sources, and so may waste more time conducting an experiment for which the results are already known. While this is obviously an undesirable environment to conduct research in, it is the current state of genomics research.

One of the key challenges for bioinformatics, as described in more detail in Section 2, is to address the problem of data accessibility in a realistic environment. Significant research is attempting to provide solutions to all or part of this problem, with several projects briefly discussed in Section 3. The DataFoundry project, outlined in Section 4, is an ongoing research effort that demonstrates how the application of meta-data to this problem can yield realistic solutions. Section 5 describes our experiences using and extending DataFoundry over the past year. Finally, we conclude with the current status of the project and a look ahead at the remaining challenges.

2. An Overview of Bioinformatics

Bioinformatics has received increased publicity over the past few years, in large part due to its importance to the Human Genome Project. At the same time, however, this term has come to mean different things to different people, ranging from atomic-level modeling of protein structures to tele-medicine. Within the context of this paper, we restrict bioinformatics to its traditional definition, meaning the management of genomics data. This is still a huge domain, encompassing laboratory information management systems (LIMS), local production databases, databases provided as community resources, and dissemination of data via email, ftp, and the web. As such, there are several significant challenges that have not yet been addressed satisfactorily, the most important of which is providing scientists effective, efficient access to all relevant data.

Understanding this challenge, and why it is interesting from a research perspective, requires an understanding of the real-world environment in which bioinformatics solutions must function. In the genomics community, several dozen large, community data sources provide a wealth of information on a specific sub-domain (e.g. human protein sequence, human protein structure, mouse DNA). In addition, several hundred smaller data sources provide highly specialized information, in many cases limited to data produced by a specific lab. While some of the information contained in these sites is duplicated in the community data sources, usually there is additional information unique to that source. Each of these sites uses a custom user interface providing vastly different query capabilities.

Because each source uses its own format, navigating between sources and transferring data between interfaces is usually more complicated than a simple mouse click or cut and paste operation. For example, a protein sequence may be represented in three-character code in one source, while another source expects it in one-character format. These translations are not necessarily difficult, but they are time-consuming when repeatedly applied. To help move between sites, some sources provide cross-references to related data located elsewhere. While this is a big step in helping identify related data, as implemented, it is far from the perfect solution. In particular, these cross-references are not consistent across databases. For example, PDB (a protein structure database) [4] and SWISS-PROT (a protein sequence database [1] contain cross-references. However, a PDB entry may reference a SWISS-PROT entry that does not reference it, but may instead reference other PDB entries. These consistency violations mean that (1) these cross-references may be treated as hints but should not be taken as definitive and (2) these links are uni-directional instead of the expected bi-directional associations.

The large number of sources, and the difficulty moving between them, is creating a crisis. The situation is made worse by technological advances that allow data to be created from the wet-lab at an ever-increasing rate, and the growing need to combine this data in new and interesting ways. Consider that while the entire genomic sequence of a small organism can be obtained in a single day [16], making sense of it requires more than simply listing the DNA data produced in the original experiment. Information — such as homologs, annotations, structures, maps, locations, species similarities, and much more — must be analyzed before the impact each gene has on the overall organism can be understood. Research of this type, often referred to as functional genomics or proteomics, is quickly becoming the focal point of most genomics and pharmaceutical efforts. Unfortunately, the data required to perform this research is spread among dozens (if not hundreds) of sources. As a result, scientists are spending an increasing amount of time performing basic data management tasks instead of pursuing their research.

On the surface, this environment sounds similar to that of a large multi-national corporation in which local offices maintain their data in their own formats. Since important decisions are made at the highest levels of the company, managers need a way to combine these sources into a consistent view upon which to base their decisions. Industry has successfully applied several technologies to addressing this problem including data warehouses, data marts, and federated databases systems. The resulting database contains all of the required information from the sources, and is able to handle queries and to generate reports based on this information. Thus, it might be expected that directly applying these approaches in a bioinformatics setting would provide similar successes. Unfortunately, this is not the case. While some of the challenges facing bioinformatics are the same as those facing industry (e.g. heterogeneous databases), unique constraints make a direct technology transfer to active scientific disciplines infeasible.

Real-world bioinformatics solutions are applied in a very active scientific domain. In this environment, the underlying terminology and technology are both constantly evolving. This means that a bioinformatics database must also continually evolve to represent the new types of information, while maintaining access to the historical data. Furthermore, the terminology used by scientists is not consistent between different sub-communities —or sometimes even within a community. For example, determining whether two sequences are homologous often involves a detailed, and very subjective, comparison between them. These distinctions make it very difficult to understand the true semantic correspondences between data from different sources, and very easy to define overly simplistic ones. Complicating efforts to agree on standards is that data sources are distributed among groups spread across the world, without any all-encompassing authority to encourage (or force) the data providers to agree on issues such as semantics, formats, etc. To date, the primary motivation for unification comes from the users, and they have not been a significant enough political force to define and enforce standards — in part because they are a divided community with different goals and requirements. Even if these problems could be resolved, three issues still persist in making bioinformatics challenging from a data management perspective. First, the persons responsible for distributing the data have historically lacked a background in data management. As a result, the data sources have evolved from flat-file and Excel spreadsheet formats, rather than from relational databases, with the expected consistency problems as a result. Second, the data does not follow well-defined rules. It appears as though every rule has exceptions, although the exception may not be known at design time, so using integrity checks, foreign keys, and other common database techniques for ensuring high-quality, consistent data is very difficult. Third, users must view and interact with the data in several ways, depending on what they want to do with it. This precludes using only summary data, and results in longer query execution time, and more complex schema and interfaces.

Unfortunately, research-driven approaches to bioinformatics often ignore these critical problems, choosing to focus on an abstract and elegant solution. As a result, these “solutions“ do not work in the field. By considering these challenges, the research becomes significantly more difficult, but the results are then applicable to real-world problems. To meet these challenges, research efforts must yield a solution that is both flexible and scalable. As the projects discussed in the next section show, that is a formidable requirement, not easily achieved.

3. Other Approaches

The Object Protocol Model (OPM) [5], developed at Lawrence Berkeley National Laboratory and licensed by Gene Logic Corp., provides a set of tools for scientific data management. In place of a traditional, relational data model, OPM uses an object-oriented data model and defines a *protocol* class for modeling experiments. The OPM tool-set consists of a schema editor, for specifying and managing OPM schemata; a graphical querying and browsing tool; and tools for conversion between relational and OPM schema so OPM can access existing scientific databases. OPM uses a federated approach, whereby all queries are formed over a global schema and passed to relevant sources after being converted to the source format. Initially, OPM required data sources use its data model before they could be included in the global schema, however, newer versions use the conversion tools to translate between the source and OPM representations. Unfortunately, the mediators that perform this transformation must be manually created and are sensitive to changes in the schema, interface and underlying data format. As a result, OPM is not sufficiently scalable or flexible to support large-scale proteomics research.

The CPL/Kleisli project (U Penn) [10] also provides tools to manage the transformation of data between databases, and to provide integrated access to multiple data sources. The transformation and integrated access is achieved through constraints and rules specified in a special purpose collection programming language (CPL). Kleisli follows a multi-database approach to accessing data across multiple data sources and does not provide an integrated schema. Hence, users are required to directly specify the rules and constraints involved in queries when accessing the databases. While extremely general, this approach prevents casual users, who may be unfamiliar with the details of the individual data sources, from fully utilizing the resource, and it does not necessarily provide a semantically consistent view of the data being returned. This approach also limits the system’s flexibility and scalability. In particular, because stored queries rely directly on the underlying source formats, they must be manually updated when the associated source schemata change. Furthermore, when a new source is to be added to the system, a new driver may need to be written by hand (depending on the source format), and existing queries must be explicitly updated to return data from that source. These tasks require a person who is intimately familiar with the semantics of the new source, the queries being updated, and all of the associated sources.

More recently, approaches have started to utilize meta-data in a variety of ways. For example, TAMBIS [2] is a federated database project that makes heavy use of meta-data in the form of ontologies. These ontologies are used to drive the user interface and query capability. The user creates a query on the global ontology, which is translated into CPL (the same language as used by Kleisli) and sent to the wrappers. The wrappers interact with the sources directly and translate data and queries between the TAMBIS and source representations. The advantages of this approach over Kleisli are that (1) the user need not be familiar with the sources, only the ontology, and (2) users automatically obtain access to new sources when they are integrated. Since the wrappers are manually created, however, this approach will not scale to hundreds of sources.

GAME [17] is part of the bio-XML open-source initiative. It is focused on creating several small XML dtds (schemata) capable of representing genetic data. Currently, it is developing a dtd for representing features of sequences – including computationally derived features. This will allow applications to easily store and exchange data, such as blast query results, in an application independent format. The long-term goal is to create a collection of dtds that can be combined as needed to represent complex genomic data and relationships. XML was chosen as the data interchange format because is a well-defined, self-describing, standard that allows information to be easily passed between applications. As XML is a standard, the project is able to leverage XML components such as parsers and style sheets from other efforts. While not a database project, GAME’s efforts in standardizing a meta-data representation are an important step in addressing current bioinformatics challenges.

On the industrial side, bioinformatics companies such as Cimmaron [6], NuTec [19], and eBioinformatics [12], have formed in recent years to address different components of the bioinformatics market. Companies such as Cimmaron focus on managing and tracking genomics data generated on site, and develop exceptional LIMS systems as a result. NuTech is one of the companies addressing data integration needs for large customers, such as pharmaceutical companies, that import data from public sources to their local sites. While requiring additional storage, this approach improves query response times and prevents competitors from covertly identifying which sequences interest them. Companies such as eBioinformatics are attempting to meet the integration challenges of smaller organizations and academics by providing a web resource that integrates several commonly used tools and data sources. While some of these companies are using automated or meta-data driven approaches to reduce their workload, most are using various brute force techniques — often just throwing a large number of programmers at the problem. Unfortunately, because these approaches typically provide solutions to very specific problems, and require significant time and resource investments to change, they are unlikely to meet the evolving needs of proteomics.

4. The DataFoundry Approach

Both academic and industrial projects have repeatedly demonstrated that, while the initial cost of developing an usable interface is high, it is dwarfed by the ongoing costs of maintaining and extending it. With this in mind, DataFoundry has focused on developing a flexible environment that reduces the costs of adding new sources and adapting to changes in previously integrated ones. The result of this effort is a typical data warehouse architecture that is supported by a meta-data driven infrastructure. The meta-data declaratively represents information that has traditionally been represented procedurally. Because the meta-data is easier to update and extend than the code it replaced is, this results in a more flexible and scaleable application. The remainder of this section provides an overview of the DataFoundry infrastructure, and the next describes some of our experiences using it.

DataFoundry is based on a mediated data warehouse architecture similar to that shown in Figure 1. This architecture promotes two views of warehouse interactions, from the top (a user performing queries) and from the bottom (data being loaded from the source). Users interact with the warehouse through a web-based GUI that provides a set of menu commands to let them do familiar tasks against data from multiple sources simultaneously. These commands hide the underlying query complexity from the users, allowing them to focus on research. Since the desired tasks may be computational, as well as descriptive, the interface is capable of transferring data from the warehouse to stand-alone programs — providing a consistent, seamless interaction to the user. We use a Java backend with a JDBC connection to translate the user commands into the appropriate SQL queries, parse the results, pass them to an external program (if required), and return the results to the user. From the other perspective, data is retrieved from the source by the wrapper, parsed into an internal representation, and passed to the mediator. The mediator converts the data from its source format to the warehouse format and stores it in the warehouse.

To make this architecture feasible in the genomics environment, the cost of creating and maintaining the wrappers and mediators must be minimal. If a source is relational, a single wrapper can easily obtain data from multiple sources through SQL commands. If the source uses flat files, as is often the case in bioinformatics, the wrapper must parse the files to obtain the data. At this time, parsers must be custom written for each flat-file format. Once the data has been read into an internal representation, it is converted into the warehouse format. This is usually a simple conversion between equivalent formats, but may require additional resources. For example, while converting between feet and meters is a simple operation, converting U.S. dollars to Yen requires information about the current rate of exchange. Because of this inherent complexity, translation programs must be also hand-written. Finally, the data is entered into the warehouse through a series of library calls. The mediator code, including both the data entry code and calls to the translation methods, is automatically generated from meta-data. As shown in the next section, this has resulted in significant reductions in the effort required to integrate new sources.

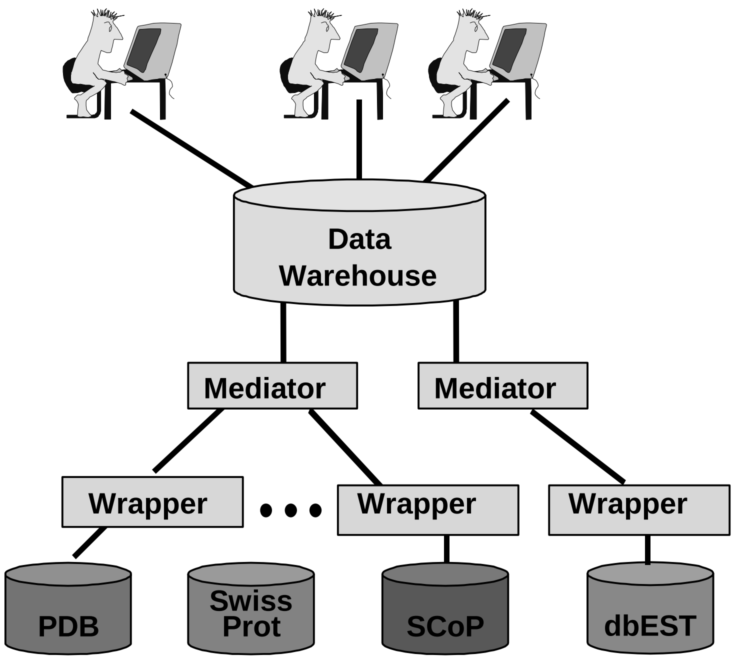

Figure 1 DataFoundry Architecture

DataFoundry uses meta-data to describe four concepts: *abstractions*, *transformations*, *schema*, and *mappings*. Abstractions define a class hierarchy reflecting the domain-specific concepts contained in the data sources. They include attributes representing different source formats of conceptually the same attribute, and, unlike relational attributes, they may be complex and multi-valued. For example, the *chain* abstraction has attributes length, name, one-character sequence, and three-character sequence, where the sequence attributes are alternative representations of the same conceptual information. Transformations define the set of legal, uni-directional translations between related attributes — for example from the one-character sequence to the three-character sequence formats, and back. The schema meta-data mirrors the target database (i.e. data warehouse) schema using an extended relational format. This meta-data is easily obtained from commercial RDBMSs, but has been extended to allow both complex object definitions and comments to be included. This allows mappings from the warehouse back to the source, an important capability if sources are ever to be queried dynamically. Where appropriate, mappings define correspondences between abstraction attributes and schema attributes. Currently, the meta-data can be represented using either Ontolingua [15], or XML [14]. We expect the XML representation to dominate as it allows leveraging the generic resources, tools and interpreters developed by the XML community.

We have implemented a mediator generator program that, when given a complete set of meta-data, outputs both a mediator class and a translation library. The library defines a C++ class hierarchy mirroring the abstraction hierarchy defined in the meta-data, but extended to include the appropriate constructors, destructors, translation, get, and put methods. When an abstraction defines a multi-valued attribute, the corresponding put method uses a linked list construct to mimic this capability. The get methods determine whether the attribute is undefined and, if it is, will attempt to obtain its value from one of the other attributes, based on the transformations defined in the meta-data. This attempt may trigger additional translations if the required attribute(s) is also undefined. Recursive cycles are detected and prevented. The mediator class defines methods that take high-level objects (e.g. instances of root classes) and enters them into the database. This requires performing transformations when appropriate, iterating over instances of multi-valued attributes, and entering the data into the warehouse only if all of the required information has been provided. More information about the DataFoundry architecture can be found in [9], and the meta-data format and mediator generator are described in [7] and [8].

5. Experiences Using DataFoundry

The mediator generator was completed over the summer of 1998 and the DataFoundry warehouse was released to LLNL users January 1999, after completion of a prototype interface. As part of our initial evaluation of the mediator generator, we compared the time required to integrate a new data source (the SCoP [18] structural taxonomy database) into a warehouse that contained data from SWISS-PROT and PDB using both the traditional, hand-coded approach and the new meta-data driven one. As Figure 2 shows, the meta-data based approach provided significant time-savings over the traditional one. While most of this saving came from the automatic generation of the mediator itself, it is interesting to notice that we also

Activity	Manual	Meta-data driven	Diff	% diff
Understanding SCoP	2.0	2.0	0.0	0
Writing wrapper	4.5	2.5	2.0	44%
Modifying schema	0.5	.05	0.0	0
Writing mediator	4.0	0.0	4.0	---
Modifying meta-data	0.0	1.0	(1.0)	---
Total time (days)	11.0	6.0	5.0	45%

Figure 2 Comparison of Approaches

achieved considerable savings writing the wrapper. The reason for this was our ability to use the class library that was automatically created for the mediator as the internal wrapper representation — so we did not spend time creating a specialized data structure for that purpose. In this light, defining the meta-data can be viewed as an alternative to creating the classes required for the wrapper. In addition, since the meta-data is defined at a higher level of abstraction, we are able to define all of the meta-data in less time than explicitly coding the classes would have taken. It is important to note that while there was considerable time-savings in this example, for sufficiently complicated data sources, the time required to understand the data and modify the schema would dominate the integration time. This does not, however, diminish the impact of the meta-data driven approach because maintaining the mediator code is reduced to the simpler task of maintaining the meta-data, reducing the overall maintenance cost.

Our experiences maintaining and upgrading the mediator generator, data warehouse, and interface over the last year and a half have highlighted the importance of designing bioinformatics systems for flexibility. The remainder of this section describes several upgrades we have performed. While none of these modifications were overwhelming, and most of our solutions were derived from standard software engineering approaches, each case highlights a type of evolution that we did not anticipate. As a result, each of them required extending our prototype — even though the initial system was designed with flexibility as a primary concern. . In a dynamic environment, even more than in general, it is critical to learn from the past. By using each of these upgrades as an opportunity to increase the flexibility of the system, not just address the current problem, we are reducing the cost of future modifications and creating a more robust infrastructure.

The most obvious code evolution has been in our interface. We started with a forms-based interface, driven by Perl CGI scripts, which allowed the user to perform a limited set of queries and returned the results in a table. This interface was easily developed and provided a way for users to interact with the warehouse without requiring them to understand the underlying database. While sufficient for simple queries, this interface did not allow users to perform the complex, iterative discovery that they needed to perform their research. We have replaced this interface with an applet/servlet interface that allows both batch and iterative queries to be formed. This interface has been designed to allow external programs to be used in processing user commands. The underlying Java code has undergone one significant revision since its initial conception to convert it from proof-of-principle prototype to a more flexible architecture. Under this new architecture, the buttons and menus containing the commands available to the user are dynamically created based on a specialized class hierarchy.

We have made several illustrative changes to the mediator generator (MG); a fundamental underlying assumption about the database schema was changed; an XML-based meta-data representation was added; and the machine and database platforms changed. Each of these modifications was simplified by initial design decisions favoring flexibility, but each still required more reworking than if the change had been initially anticipated. The remainder of this section describes these changes, highlights the design features that simplified their implementation, and outlines how the infrastructure was updated to reduce the impact of future changes.

When we integrated our fourth source, dbEST [3], we changed the initial assumption that only one database schema would be associated with all of the data. This decision was made because combining radically different types of data into the same schema would result in wasted space, increased response time, or both. For example, consider that the available information about people (authors, submitters, etc.) varies dramatically between sources. If a single table contained the information about all people referenced by any source, it would contain fields for information present in any of the sources. Furthermore, queries requiring information about people would be made against that large table, even if we knew which source referenced the person. However, if there were several tables, each could contain the subset of data provided by a particular set of sources without wasting space on those fields not provided. Since the dbEST data was significantly different from the data already contained in the warehouse, we created a second database to contain it. Of course, this made retrieving data common to both sources significantly more difficult, since multiple tables were queried to generate a response. Fortunately, these queries are not common. To properly map data to these distinct schemata, we created a mediator for each target database. This was a dramatic departure from our initial belief that there would be a single mediator for the entire warehouse. Updating the MG required adding arguments for the location of the mediator-specific meta-data (the abstraction data is still shared, but the schema and mappings are not), and ensuring the resulting mediator code was aware of the use of multiple mediators (e.g. the library could not assume a specific target). This rewriting was simplified by the object-oriented design of the MG, and the initial use of a parameter file to define configuration variables.

When the mediator generator was first written, XML was not ubiquitous, as it is today. We saw few obvious choices as to the "best" data representation format. At the time, we were considering pursuing automatic wrapper creation research, so we chose to represent the meta-data using a knowledge-reasoning language developed at Stanford, Ontolingua. Since we chose not to pursue that line of research, and XML has come to dominate the data interchange world, we have extended the MG to parse an XML representation, and have written a script to translate the Ontolingua representation to the XML one. Again, extending the mediator generator was aided by the object-oriented design that allowed us to easily identify the parsing methods, create a new parser parent class, and use subclassing to override functionality as needed. If this evolution had been originally envisioned, the parsing methods would have been associated with a single class instead of distributed as readFromFile methods in multiple classes.

Changing machines was a little more work. We had initially developed the system as a prototype, therefore, much of the information was hard-coded. Constants such as the name of the database and server, accounts, passwords, directories where specific scripts are kept, etc., were spread throughout the infrastructure. Obviously, almost all of this information became obsolete when the platform changed. Our solution was to create configure files, one for each language, that define named constants to represent all of the literal information required by the program, and to update the files to refer to them instead of the literals. Now, when we move platforms, only these files need to be updated and recompiled to reconfigure the infrastructure for the new environment. Obviously, if we had designed the system with platform portability in mind originally, the initial move would have been greatly simplified.

Finally, we are extending the code base to work with Oracle as well as Sybase. This has been the most challenging of the upgrades to date. The modular design and use of standard SQL in most of the infrastructure has made it relatively easy to swap out one connection library for another. For example, the Java JDBC interface simplified the interface extension, and we used two simple libraries that export the same interface but convert the calls to either Sybase or Oracle calls as appropriate to achieve similar results for C/C++. However, for the perl scripts, wrapper scripts had to be created to call either Sybase or Oracle perl interpreters based on the value of an environment variable. An unexpected result of the migration was the need to modify the database schemata to remove conflicts in column names and index names— even though Sybase had accepted them. Addressing this problem required rewriting queries to refer to new column names, and making each index name globally unique. A more portable approach would have been to avoid using names that could have conflicted, for example, by prefixing column names with DF_, and index names with the name of the associated table.

6. Current status and Future Directions

At this time, the data warehouse contains approximately 40 GB of data from four external, public data sources. It is updated weekly, and used by a group of local biologists. We are actively extending the interface capabilities to include additional commands that further enhance our users ability to utilize the data. For example, the ability to email a data set in an Excel-importable format was recently added based on user requests, allowing scientists to easily include the results in papers. We are also investigating third-party XML tools to use as a graphical interface to our meta-data, as a validating GUI would reduce the number of syntax errors in the meta-data, allow for easier manipulation of the data, and, we hope, provide a way to intuitively present the meta-data to others.

While we were developing the MG and associated infrastructure, the internet explosion hit the genomics domain. When DataFoundry started, approximately 30 publicly available genomics data sources existed, now there are over 500 [11]. While our current approach is expected to scale to the order of dozens of data sources, it will not scale to hundreds. To meet this new challenge, we are investigating a hybrid architecture. In this architecture, critical data sources are fully integrated into the warehouse, so they are available in a semantically consistent format, and other data sources are dynamically accessed through special commands from the interface. Semantically reconciling this non-integrated data would be the responsibility of the scientist making the query, making this far from an ideal solution. However, for now we believe this hybrid approach is the only realistic way to provide access to the large number of data sources required by functional genomics.

Successfully implementing this architecture will require dramatically extending our use of meta-data in two ways. First, we will represent a basic ontology of the genomics domain that will allow us to (roughly) identify and categorize web pages as we encounter them. We expect to heavily leverage the existing work on genomics ontologies for this step. Second, we will define and use a data format capable of describing all of the significant features of both the query and results pages in order to successfully wrap the source. We have developed an RDF [20]schema we believe is capable of describing these pages. To associate instances of this schema with various sources, we are taking a two-pronged approach. First, we will use the domain ontology in conjunction with extended soft-bot technology [13], to generate coarse descriptions of web sources. Because this will not give us as much detailed information as we want, we will also encourage database providers to publish the capabilities of their interface using this format. These instances will be used by a wrapper generation program to create wrappers capable of querying and returning data from the corresponding sources. The query engine will also use the meta-data to identify the set of pages that should be queried to return a specific set of information. Failure to fully utilize meta-data will make addressing this problem significantly more difficult, if not impossible.

7. Conclusions

Bioinformatics is plagued with information overload. The underlying science and supporting technology is changing at a rapid pace, forcing terminology, formats, and interfaces to play a constant game of catch-up. The result is data distributed among hundreds of sources, each using custom formats and inconsistent terminologies. As the focus of the genomics community moves from sequencing to proteomics, the need to provide reliable access to as much information as possible is increasing. Semantic integration of the community resources has long been a goal of bioinformatics. With the growing importance of the smaller sources, however, even achieving that lofty goal will not be sufficient to meet the growing demand for data accessibility. What is needed in the long-term is a method to automatically identify, categorize, and support interactions with genomics sources. This paper has used the DataFoundry project to highlight different applications of meta-data to bioinformatics, and to demonstrate the critical role meta-data will play in addressing the meeting future needs of this community.

References

[1] A. Bairoch and R. Apweiler. “The SWISS-PROT protein sequence database and its new supplement TrEMBL.” *Nucleic Acids Res.* 24:21-25(1996).

[2] P. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, R. Stevens. “TAMBIS – Transparent Access to Multiple Bioinformatics Information Sources.” *In Proceedings of the Sixth International Conference on Intelligent Systems For Molecular Biology* ISMB-98. Montreal. 1998.

[3] M.S. Boguski, T.M. Lowe, and C.M. Tolstoshev. “dbEST – database for expressed sequence tags.” *Nat. Genet*. 4(4):332-3. Aug 1993.

[4] J. Callaway, M. Cummings, B. Deroski, P. Esposito, A. Forman, P. Langdon, M. Libeson, J. McCarthy, J. Sikora, D. Xue, E. Abola, F. Bernstein, N. Manning, J. Sussman. *Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description* Version 2.1. Tech Report available from www.pdb.bnl.gov. October. 1996.

[5] A. Chen and V. Markowitz, An Overview of the Object Protocol Model (OPM) and the OPM Data Management Tools, *Information Systems*, Vol. 20, No. 5, 1995.

[6] Cimarron Software, Inc. Salt Lake City. UT. www.cimsoft.com

[7] T. Critchlow, M.Ganesh, R. Musick, “Automatic Generation of Warehouse Mediators using an Ontology Engine.” *Proceedings of the 5th International Workshop on Knowledge Representation Meets Databases* (KRDB). May 1998.

[8] T. Critchlow, M.Ganesh, R. Musick. “Meta-Data Based Mediator Generation. *Proceedings of the 3rd IFCIS Conference on Cooperative Information Systems* (CoopIS). Aug 1998.

[9] T. Critchlow, K. Fidelis, M. Ganesh, R. Musick, T. Slezak. “DataFoundry: Information Management for Scientific Data.” *IEEE Transactions on Information Technology in Biomedicine* Volume 4. Number 1. March 2000. pg 52-57

[10] S. Davidson, C. Overton, V. Tannen. “BioKleilsi: A Digital Library for Biomedical Researchers”. *Journal of Digital Libraries*. Nov. 1997.

[11] DBCAT, The Public Catalog of Databases. http://www.infobiogen.fr/services/dbcat/

[12] eBioinformatics. Eveleigh Australia. www.eBioinformatics.com

[13] O. Etzioni, D. Weld. “A Softbot-Based Interface to the Internet.” *CACM*, July 1994.

[14] Extensible Markup Language (XML) 1.0 W3C Recommendation. Tech Report REC-xml-19980210 available at http://www.w3.org/TR/REC-xml

[15] A. Farquhar, R. Fikes, & J. Rice. *The Ontolingua Server: A Tool for Collaborative Ontology Construction.* Tech Report. Knowledge Systems Laboratory, 1996.
[16] Danger bacterium decoded in a day. Press Release. DOE Joint Genome Institute. May 9.
[17] S. E. Lewis, E. Frise. *GAME (Genome Annotation Markup Elements)* Version 0.3. Tech Report. Available from www.bioxml.org.
[18] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. “SCoP: a structural classification of proteins database for the investigation of sequences and structures.” *Journal of Molecular Biology*. 1995. 247:536-540.
[19] NuTec Services. Stafford TX. www.nutecservices.com
[20] *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. Tech Report REC-rdf-syntax-19990222 available at http://www.w3.org/TR/REC-rdf-syntax/

Work performed under the auspices of the U.S. DOE by LLNL under contract No. W-7405-ENG-48 . UCRL-JC-139907